Evaluation of an Object-Based Data Model Implemented Over a Proprietary, Legacy Data Model

Daniel L. Pollard, Joseph W. Hales, PhD
Division of Medical Informatics, Duke University Medical Center, Durham, NC

Most computerized medical information today is contained in legacy systems. As vendors slowly move to open systems, legacy systems remain in use and contain valuable information. This paper evaluates the use of an object model imposed on an existing database to improve the ease with which data can be accessed. This study demonstrates that data elements can be retrieved without specific programming knowledge of the underlying data structure. It also suggests that underlying data structures can be changed without updating application code. Programs written using the object model were easier to program but ran greater than one order of magnitude slower than traditionally coded programs. In this paper, the legacy information system is introduced, the methods used to implement and evaluate the object-based data model are explained, and the results and conclusions are presented.

INTRODUCTION

TMR (The Medical Record)[1] is a longitudinal computer-based patient record which has been developed at Duke University over the last 20 years. It is in use in eight sites at Duke University Medical Center and in ten sites outside of Duke. Like many legacy systems, TMR is written in a proprietary language (GEMISCH)[2,3] and uses a proprietary database system.

We are in the midst of re-engineering TMR to move from its proprietary database management system (DBMS) to an off-the-shelf DBMS. Once the transition is complete, we expect to be able to use a standardized tool such as SQL to perform the majority of our queries. But until the massive re-engineering project is finished, we need an interim solution to better access the wealth of information contained in our legacy system.

Currently, extracting information from TMR requires custom programming. General extraction methods do exist but are difficult to use or do not provide access to every data element. To produce custom reports, programmers must be taught the structure of the TMR patient record and then hard-code it into their applications. Many problems have resulted from this hardcoding of structural information into application programs: data are retrieved inconsistently due to differences in the individual coding styles of programmers; improvements in TMR which change the data structure require massive reprogramming of applications; training new programmers is expensive and time consuming.

To remedy these problems, we have created a system which automatically tags each data element in TMR with an intuitive, hierarchical identifier called an *object tag*. Programmers and users can now reference TMR data elements without knowing where and how they are stored in the patient record; they only need to know the name of the object's tag. The tagging is accomplished by extending TMR to include meta-data. By the term meta-data, we mean data about data that explicitly defines the structure of the patient record in a manner accessible to all applications.

In this paper, we describe the design, implementation and evaluation of a system to incorporate meta-data into TMR. First, the design of the system and the method of evaluation are discussed. Next, the results of the evaluation are presented. Finally the results are discussed.

METHODS AND PROCEDURES

System Design

The TMR patient record is a modular structure which has nine major sections. The sections include demographics, appointments, SOAP notes, problems, medications, studies (laboratory results), subjective and physical findings, encounters and accounting. The accounting section was selected for this project as a prototypical section. The accounting section contains over 100 individual data elements which can be represented in a hierarchical structure seven levels deep. This section was chosen for two reasons. First, the accounting section's complexity is about average compared with data found elsewhere in the patient record. Secondly, a generalized method for getting at this data does not exist.

Three steps were involved in incorporating meta-data into TMR. First, an object-based model of TMR was derived. Next, an editor was written to capture and store the meta-data. Finally, a generalized engine was programmed to interpret the meta-data. These steps are described in the following paragraphs.

Table 1. TMR Objects

Objects Types
Section
Block
Line
Field
Molecule
Atom

Table 2. TMR Atoms

Atom Types
Date
Value
TMR Code
Free Text
ID Number
Modifier
Counter
Pointer
Time
Delimiter

To define an object-based model as proposed by Hales[4], we performed an object-oriented analysis[5] of the structure of the accounting section. *Table 1* lists the different types of objects identified in the analysis. The objects were derived from the logical and physical structure of the patient record. The specific data elements found in the accounting section are called atoms in our object-based model. The types of atoms are listed in *Table 2.*

Next, we created a data structure called an *object frame*. An object frame is a representational structure containing a set of attributes[6] which describe a TMR data element. It should be noted that the object frame does not explicitly contain any methods[7] (methods as defined in the object oriented paradigm) but some of the attributes stored in the frame do provide the information needed to link object with the appropriate methods. An object frame is defined for each data element present in the TMR accounting section. The attributes which constitute an object frame are listed and described in *Table 3.*

An *object frame definition editor* was designed to capture efficiently the attributes of each TMR object and specify the relationships between the objects. The editor was implemented using an off-the-shelf relational database (Microsoft Access 2.0). In addition to capturing an object's attributes, the editor automatically generates an *object tag* for each object. The object tag is a hierarchical name which uniquely identifies each object. The object's tag is generated by starting with the object's name and pre-pending the name of the parent object, and then pre-pending the name of the parent's parent, etc. until the root object's name is pre-pended. In this study, the root object is the accounting section and it has a the name: ACC.

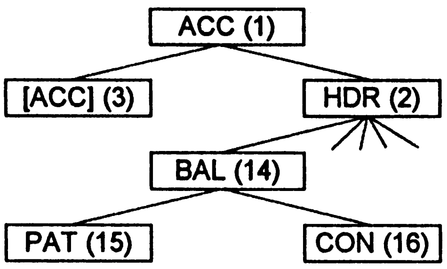
Each tag can be alternatively represented by using the ID numbers of each object in the lineage. This form is called the *dot notation*.

Table 3. Attributes of an Object Frame and an Example

Attribute Name	**Description**	**Example**
Object ID	Unique ID of object	15
Object Name	Name of object	PAT
Parent ID	Object ID of parent object	14
Description	Description of the TMR data element	Patient responsible balance owed
Offset	Logical offset of data element within the TMR record structure	1
Object Type	Type of object (see *Table 1*)	atom
Atom Type	Type of atom (see *Table 2*)	value
Delimiter	Type of delimiter used to separate like objects	#
Substructure	Sub-object (children)	none

The following example illustrates how both the object tag and dot notation are derived from the hierarchical structure of the objects. The example calculates the object tag and dot notation for a the *patient responsible balance owed (PAT)* object which is defined in the object frame example in *Table 3.*

Figure 1. Hierarchical Structure of Selected Accounting Objects



The boxes in *Figure 1* represent objects and are labeled with the attribute Object Name. The number in parenthesis is the Object ID. The lines represent the parent/child relationship between the different objects. Starting with a target object (PAT) and following the hierarchy up to the root (ACC) its object tag and dot notation can be defined. For example, the object tag for PAT is ACC.HDR.BAL.PAT and the corresponding dot notation is 1.2.14.15. The object tag for BAL is ACC.HDR.BAL while the dot notation is 1.2.14.

The frame editor generates two outputs. The first is a table of all the object frames and their corresponding attributes. The second is a table which translates the object tags to their dot notation. These two tables were exported into a format readable by the TMR system. These files created on the TMR system constitute the first half of the *meta-dictionary.*

The second half of the meta-dictionary is the *dot notation resolution engine.* The engine was written to resolve object tags into their corresponding dot notation and to retrieve the specified data element from the patient record. It is important to note the general nature of this engine and the fact that it only uses the information contained within the meta-dictionary to retrieve the patient data. In object oriented terminology, the engine contains a generalized set of methods which it assigns to the objects.

Physically, the engine is written in TMR's proprietary language GEMISCH and is run as a subroutine which can be attached to any GEMISCH application. The engine has a common interface from which it receives the object tag of the desired data element. It then returns the referenced data element from the current patient record. The engine can interpret all of the data structures present in the accounting section. Extending this engine to include all of TMR's data structures is quite feasible.

System Evaluation

Three financial reports, in increasing order of complexity, were devised to evaluate the ease of use and performance of the dot notation resolution engine.

- REPORT1 lists each patient's name and outstanding balance.
- REPORT2 generates a report of each patient's most recent charge.
- REPORT3 generates a detailed report of the patient payments for each of their encounters along with a total amount due.

Each report was programmed by an experienced GEMISCH programmer using two different coding methods. The first version used traditional hard coding techniques and the second version used tagged objects. Two measurements were taken for each of the six programs: 1) the time to write the portion of the program which retrieved the data from the patient record and 2) the time to execute each report for 100 patients.

RESULTS

The results from the study comparing the performance of three report generating programs are presented in *Table 4.* Two versions of each program were written: the hard coded version and the tagged object version.

These findings show that the programs using tagged objects took less time to code but performed much slower than the hard coded programs.

For all three reports, the output from the hard-coded version is identical to the output from the tagged-object version.

Table 4. Time to Code and Run the Hardcoded and Tagged Object Application Programs

Program Name	Coding Method	Coding Time (minutes)	Run Time (minutes)
REPORT1	Hard	2:00	0:04
	Tagged	2:00	3:20
REPORT2	Hard	4:00	0:05
	Tagged	4:00	5:15
REPORT3	Hard	7:00	0:05
	Tagged	5:00	6:13

DISCUSSION

The results indicate that the tagged-object reports successfully generated the same output as the hard-coded reports. Functionally, it did not matter which method was used to generate the report. The only measured differences were in application running time and in application coding time.

Application Running Time

For all of the reports, the tagged-object version ran greater than an order of magnitude slower. The slower performance of the tagged-object version can be attributed to two factors. First, the method of translation from the object tag to the dot notation is inefficient. The current implementation uses a linear search of an external file to perform this match. The hard coded program did not have to do this translation. The second factor is the implementation language of the dot notation resolution engine. It is coded in GEMISCH, the same language as the report generation programs. All of the hard coded programs' calls are implemented at the assembly language level.

The first pass at improving the efficiency of the meta-dictionary should be focused on the name resolution procedure. Replacing the linear search with either a binary search or an index should dramatically decrease the run time.

The next improvement would be to rewrite the dot notation resolution engine in lower level code (assembly language) and incorporate it into the GEMISCH language. This will have two benefits. First, the engine should run faster in assembly language. Secondly, programming using tagged objects could be streamlined. Currently, an object tag must be placed in a variable, resolved and the result placed into a new variable before it can be integrated within traditional GEMISCH programming structures. Optimally, the object tag would need no special treatment and could be embedded directly within existing programming structures.

We expect these improvements to increase the speed of execution to within the same order of magnitude as a hard coded TMR program. At this level, it becomes realistic to tradeoff a slower performance for a major simplification in programming.

Application Coding Time

The differences in coding time for an experienced GEMISCH programmer are not substantial. For data elements deep within the patient record hierarchy, the object tags reduce the amount of programmer written code need to extract them.

The major benefit to using tagged objects is expected to be seen with inexperienced and non-GEMISCH programmers (which includes most TMR users). The novice or non programmers would not have to understand the structure of the patient information and could retrieve data simply by object tag. Getting at the data now becomes much easier and requires less dependence on system programmers. Non GEMISCH programs can be written to retrieve information from the engine. TMR can be opened up to whole new groups of users including clinicians, researchers and administrators.

Other Results

The most significant result of this study is not seen in the results section. Simply the success of the tagged object programs shows that an object model can be successfully placed on top of an existing, proprietary, legacy data model. This has two major implications: 1) data elements can be retrieved without knowledge of the underlying structure, and 2) the data structures can be changed without updating application code. Derived benefits of this work include the ability to create a generalized report generator, a reduction in complexity of the application code and a reduction in application maintenance.

FUTURE DIRECTIONS:

The preliminary evaluation of the object tags presented in this paper is promising, but further evaluations should be made. Additional measurements should be made to measure the difference in time it takes to teach a non-programmer to use tagged objects vs. the time it takes to teach them the TMR data structure and its implementation in GEMISCH. Also, the retrieval speed of tagged objects should be compared with retrieval using standard query tools of relational databases[7].

After additional evaluation, future work should focus on improving the execution speed of the dot notation resolution engine. Additionally, the set of methods available to the engine should be expanded. One such method would be the ability to store tagged objects back to the patient record. Finally, the engine should be extended to include all sections of TMR.

The study has contributed to the task of identifying the structure and type of the data elements which have evolved in TMR over the last 20 years. By systematically placing an object model over TMR, we are explicitly defining the content of a well developed and proven computerized patient record.

Acknowledgments
This work was supported in part by National Library of Medicine Training Grant LM07071-3

References

[1] Stead WW, Hammond WE. Computer-based medical records: the centerpiece of TMR. MD Comput 1988;5(5):48-62.

[2] Straube MJ, Hammond WE, Stead WW. The GEMISCH programming language. In: Orthner HF, Blum BI, ed. Implementing Health Care Information Systems. New York: Springer-Verlag, 1989:384-395.

[3] Hammond WE, Stead WW. The evolution of GEMISCH and TMR. In: Orthner HF ed. Proceedings of the Tenth Annual Symposium on Computer Applications in Medical Care. New York: IEEE 1986;147-156.

[4] Hales JW. Reverse engineering objects into the TMR record structure. American Medical Informatics Association Spring Congress. San Francisco, CA. 1994;82.

[5] Coad P, Yourdon E. Object-oriented analysis. Englewood Cliffs, New Jersey: Yourdon Press 1991.

[6] Rumbaugh J, Blaha M, Premerlani W, Eddy F, Lorensen W. Object-Oriented modeling and design. Englewood Cliffs, New Jersey: Prentice Hall, 1991.

[7] Prather J, Hales JW, Lobach DF, Hage ML, Fehrs, SJ, Hammond WE. Converting a legacy system database into relational format to enhance query efficiency. Symposium on Computer Applications in Medical Care submission. April 24, 1995.